Towards Open-World Feature Extrapolation: An Inductive Graph Learning Approach

Qitian Wu, Chenxiao Yang, Junchi Yan
Department of Computer Science and Engineering
Shanghai Jiao Tong University

Background for Attribute Feature Learning

- **General problem:** learn a mapping from input features to labels
  - Input data $\mathbf{x} = [x_1, x_2, \cdots, x_d]$ where x_i denotes the i-th input feature
  - Assume a prediction model $f : \mathbf{x} \rightarrow y$ and objective

$$f^* = \arg\min_f \mathbb{E}_{(\mathbf{x},y)\sim D}[l(f(\mathbf{x}), y)]$$

- **Applications**
  - Tabular data: weather/income/usage prediction, disease diagnosis…
  - Real systems: recommendation, advertisement, question answering…

Scenario 1: Predict a person's income with aga/occ/edu

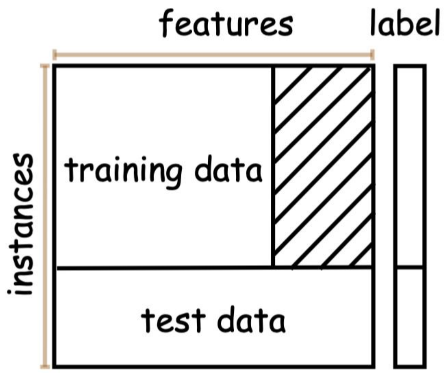
	age	occ	edu	income
o_1	x_{11}	x_{12}	x_{13}	y_1
o_2	x_{21}	x_{22}	x_{23}	y_2
o_3	x_{31}	x_{32}	x_{33}	?
$\cdots$		$\cdots$		

Scenario 2: Predict whether a user would click an item with attributes

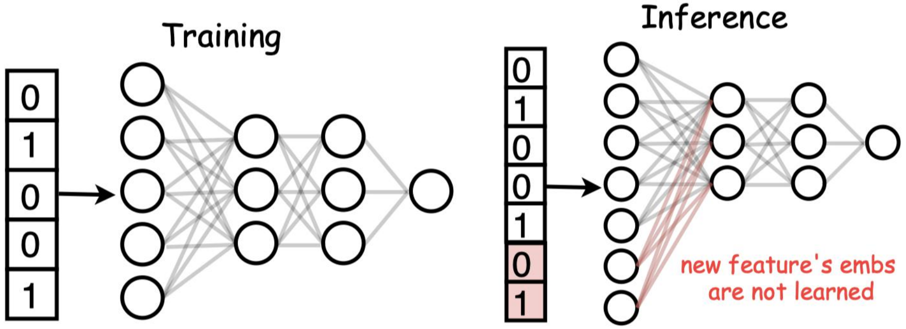
user features: age/gender…
item features: category/price…

Challenges and Limitations of Neural Networks

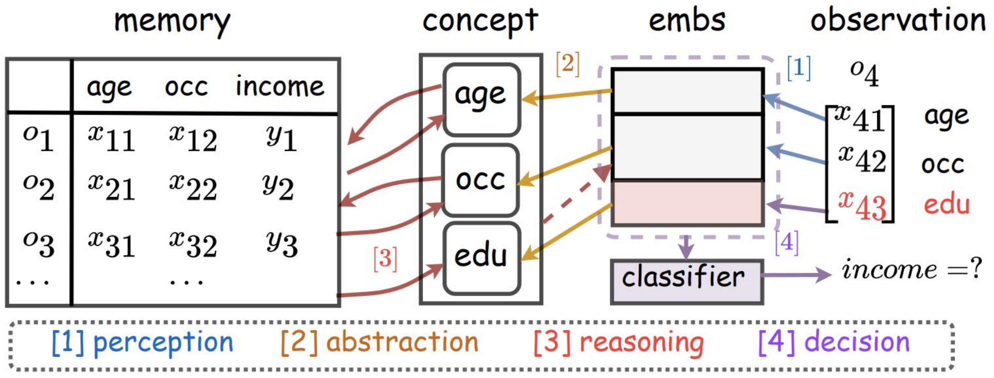
- **Challenges for attribute feature learning**
  - New features dynamically appear (unseen features in test set)
  - Scenarios: heterogeneous data sources, multi-modal data



- **How can neural networks deal with new features**
  - Retraining from scratch
    - Issue: time-consuming
  - Incremental learning on new features
    - Issue: over-fitting & catastrophic forgetting



- **Inductive reasoning ability**
  - Humans possess inherent ability for understanding new infromation



Problem Formulation

- Preprocessing: convert raw inputs to multi-hot vectors
  - Raw input $\mathbf{r}_i = [r_{i1}, r_{i2}, \cdots, r_{id}]$ where r_{im} denotes the m-th raw feature
  - For categorical feature: one-hot encoding representation
  - For continuous feature: first discretization then one-hot encoding

$$\mathbf{x}_i = [\mathbf{x}_i^1, \mathbf{x}_i^2, \cdots, \mathbf{x}_i^d] \quad \text{where } \mathbf{x}_i^m \text{ is a one-hot vector}$$

- Open-world feature extrapolation:

Training Set	domain generalization $\longrightarrow$ $\boxed{\mathcal{X}_{tr} \subset \mathcal{X}_{te}}$	Test Set
Data $\{(\mathbf{x}_i, y_i)\}_{i \in I_{tr}}$		Data $\{(\mathbf{x}_{i'}, y_{i'})\}_{i' \in I_{te}}$
Feature space $\mathbf{x}_i \in \mathcal{X}_{tr} = \{0,1\}^D$		Feature space $\mathbf{x}_{i'} \in \mathcal{X}_{te} = \{0,1\}^{D'}$
Label space $y_i \in \mathcal{Y}$		Label space $y_{i'} \in \mathcal{Y}$

- Two cases causing feature space expansion:

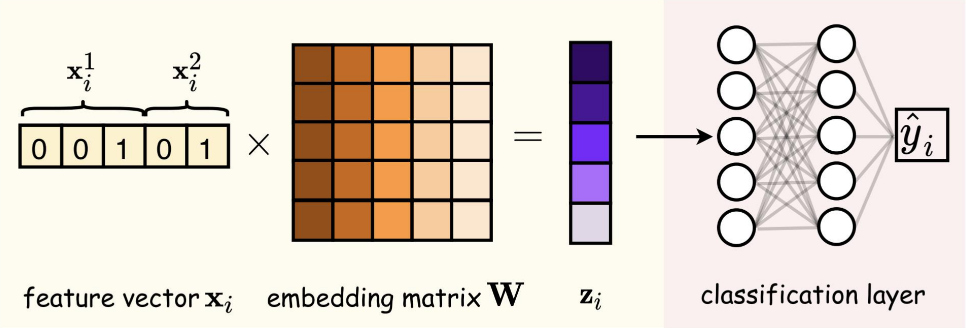
  1) new raw features come, 2) unseen feature values out of known range

Key Observation 1: Permutation-Invariance

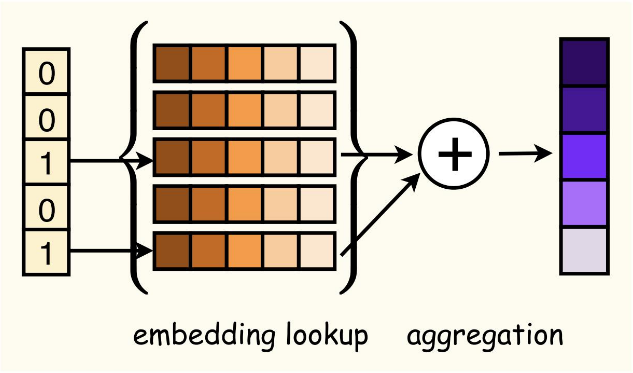
- Neural networks can be decomposed into two parts

$$\hat{y}_i = h(\mathbf{x}_i; \phi, \mathbf{W})$$

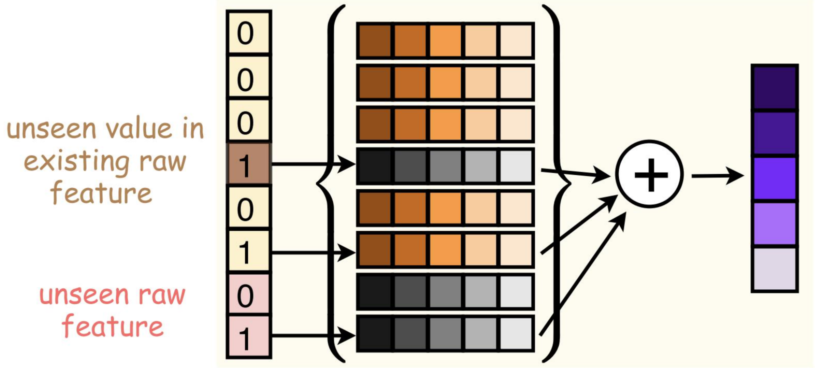
$$\Leftrightarrow \begin{cases} \mathbf{z}_i = \mathbf{W}\mathbf{x}_i \\ \hat{y}_i = \mathrm{FFN}(\mathbf{z}_i; \phi) \end{cases}$$



- Equivalent view: feature embedding look-up + embedding aggregation



Key insight:
The permutation-invariance property enables variable-length input features



Key Observation 2: Feature-Data Graph

- The input feature-data matrix can be treated as a **bipartite graph**

Input data matrix
$$\mathbf{X}_{tr} = [\mathbf{x}_i]_{i \in I_{tr}} \in \{0,1\}^{N \times D}$$

$$\begin{cases} \text{Feature nodes } F_{tr} = \{f_j\}_{j=1}^{D} \\ \text{Instance nodes } I_{tr} = \{o_i\}_{i=1}^{N} \\ \text{Adjacency matrix } \mathbf{X}_{tr} \end{cases}$$

Advantage of graph representation:

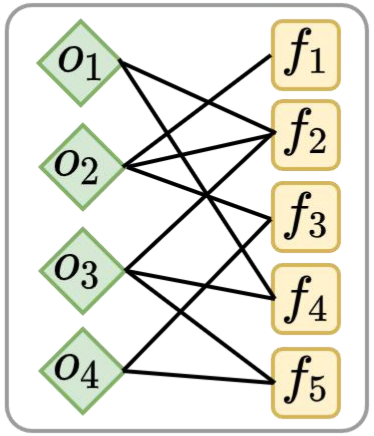
1) Variable-size for features/instances
2) Missing values are allowed

Key insight:
Convert inferring embeddings for new features to inductive representation on graphs

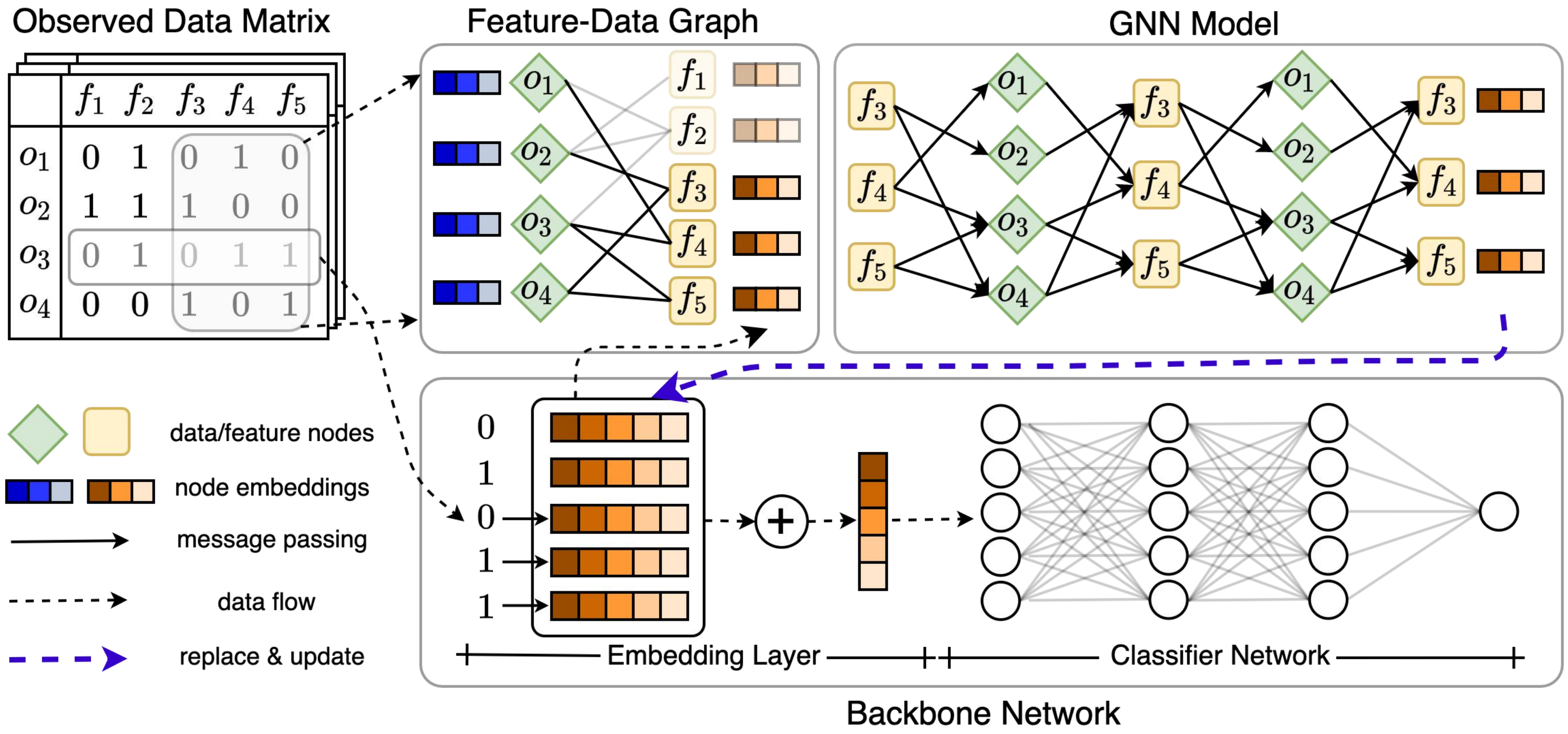
Observed Data Matrix

	f_1	f_2	f_3	f_4	f_5
o_1	0	1	0	1	0
o_2	1	1	1	0	0
o_3	0	1	0	1	1
o_4	0	0	1	0	1

Feature-Data Graph



Proposed Model Framework: FATE



- High-level GNN: take feature-data matrix as input and update feat. embeddings
- Low-level backbone: take each instance as input and output prediction

Details for Proposed Model

❑ **GNN model feedforward**

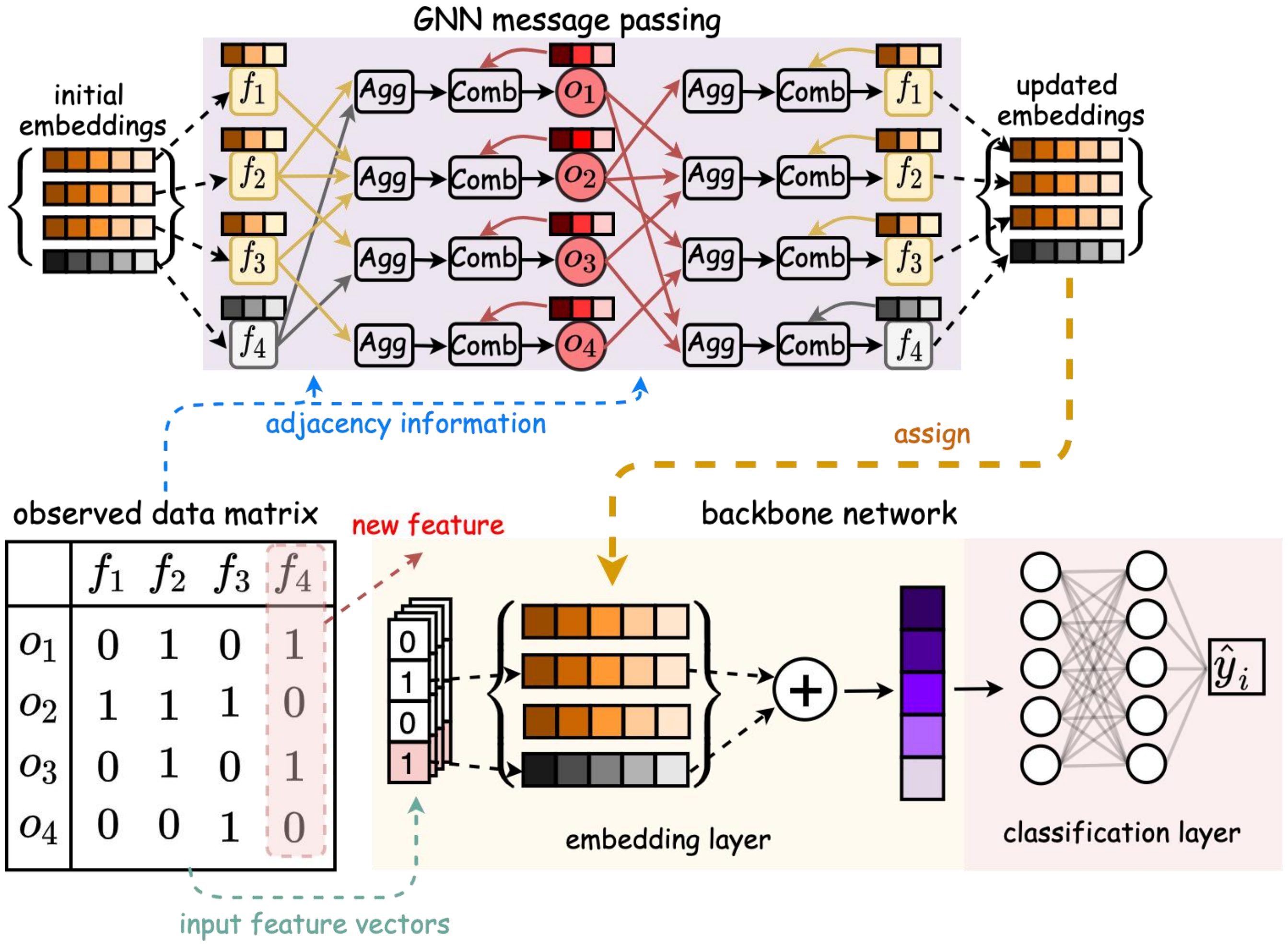
- Feature nodes $\{\mathbf{w}_j\}_{j=1}^{D}$ (initial embeddings as $\mathbf{w}_j^{(0)}$)
- Instance nodes $\{\mathbf{s}_i\}_{i=1}^{N}$ (initial states $\mathbf{s}_i^{(0)} = \mathbf{0}$)
- Message passing rule:

$$\mathbf{a}_i^{(l)} = \textsc{Agg}(\{\mathbf{w}_k^{(l-1)} | \forall k, x_{ik} = 1\})$$

$$\mathbf{s}_i^{(l)} = \mathbf{P}^{(l)}\textsc{Comb}\left(\mathbf{s}_i^{(l-1)}, \mathbf{a}_i^{(l-1)}\right)$$

$$\mathbf{b}_j^{(l)} = \textsc{Agg}(\{\mathbf{s}_k^{(l-1)} | \forall k, x_{jk} = 1\})$$

$$\mathbf{w}_j^{(l)} = \mathbf{P}^{(l)}\textsc{Comb}\left(\mathbf{w}_j^{(l-1)}, \mathbf{b}_j^{(l-1)}\right)$$



	f_1	f_2	f_3	f_4
o_1	0	1	0	1
o_2	1	1	1	0
o_3	0	1	0	1
o_4	0	0	1	0

Details for Proposed Model

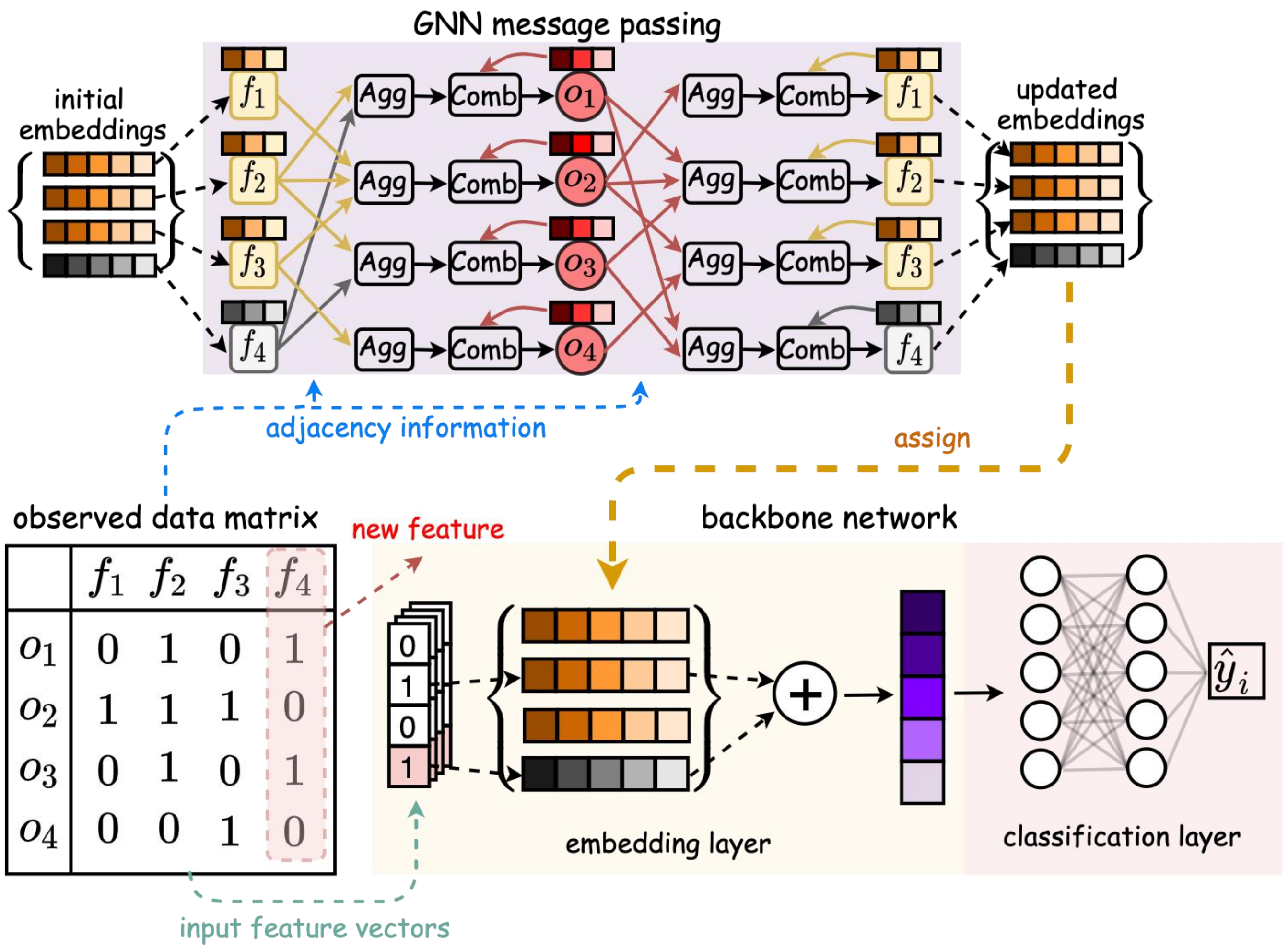
- **Entire feedforward compute**
  - Query feature embeddings
    - For old features: $\mathbf{W}$
    - For new features: set as zero
  - Updata feature embeddings

$$\hat{\mathbf{W}} = [\mathbf{w}_j^{(L)}]_{j=1}^{D} = g(\mathbf{W}, \mathbf{X}; \omega)$$

  - Assign to backbone and output predicted results

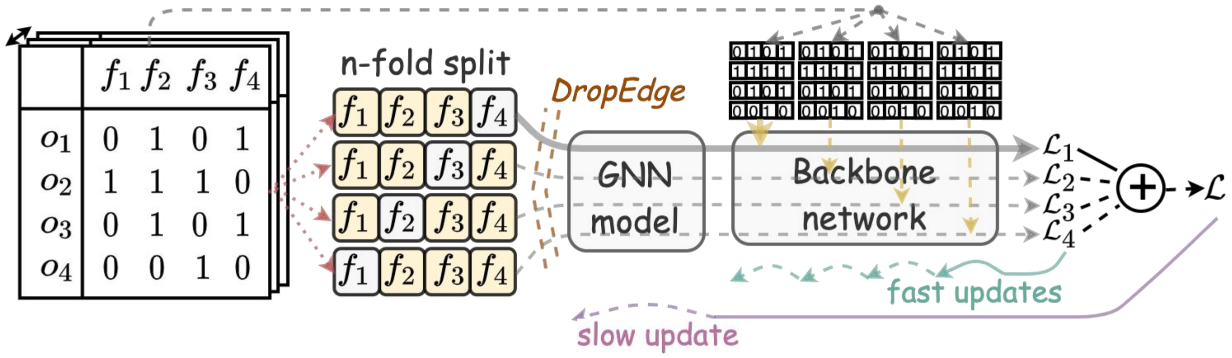
$$\hat{y}_i = h(\mathbf{x}_i; \phi, \hat{\mathbf{W}})$$

Note: 1) $\mathbf{X}$ can be either training or test data; 2) the permutation-invarance and graph representation enables arbitrarily sized $\mathbf{X}$

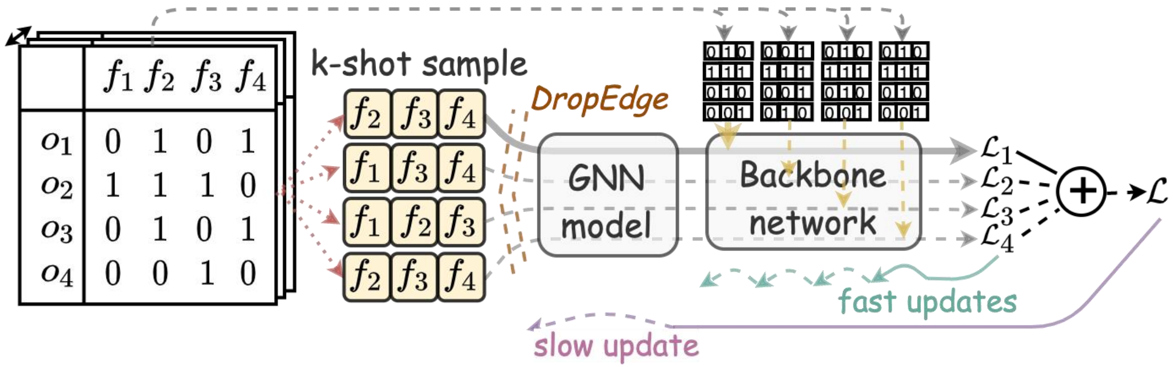


Proposed Training Approach

- Two useful techniques for **learning to extrapolate**
  - Proxy training data
    - Self-supervised learning:
      n-fold splitting input features
    - inductive learning:
      k-shot sampling input features
  - Asynchronous Updates
    - Fast/slow for backbone/GNN
- DropEdge regularization
- Scaling to **large** systems
  - Mini-batches along the instance dimension (complexity $O(Bd)$)



(a) Self-supervised learning with n-fold splitting



(b) inductive learning with k-shot sampling

Generalization Error Analysis

- **Key aspect:** we treat input data matrix as a whole and the proposed proxy data-based training approach samples data point from

$$\mathcal{S} = \{(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \cdots, (\mathbf{X}_M, Y_M)\} \quad \text{where } M \propto \mathcal{O}\left(\frac{d!}{(d-k)!k!}\right)$$

- The empirical risk over training data

$$R_{emb}(h_\mathcal{S}) = \frac{1}{M}\sum_{m=1}^{M} \mathcal{L}(Y_m, h(\mathbf{X}_m; \psi_\mathcal{S}))$$

- The generalization error can be defined as

$$R(h_\mathcal{S}) = \mathbb{E}_{(\mathbf{X},Y)}[\mathcal{L}(Y, h(\mathbf{X}; \psi_\mathcal{S}))]$$

- We care about expected generalization gap over random sampling

$$\mathbb{E}_A[R(h_\mathcal{S}) - R_{emp}(h_\mathcal{S})]$$

Generalization Error Analysis (Cont.)

- **Theorem.** Assume the loss function is bounded by $l(y_i, \hat{y}_i) \leq \lambda$. For a learning algorithm trained on data $\{\mathbf{X}_{tr}, Y_{tr}\}$ with T iterations of SGD updates, with probability at least $1-\delta$, we have

$$\mathbb{E}_A[R(h_{\mathcal{S}}) - R_{emp}(h_{\mathcal{S}})] \leq \mathcal{O}(\frac{d^T}{M}) + \left(\mathcal{O}(\frac{d^T}{M^2} + \lambda)\sqrt{\frac{\log(1/\delta)}{2M}}\right)$$

where $M \propto \mathcal{O}\left(\frac{d!}{(d-k)!k!}\right)$ and k denotes the size of sampled features

Note: 1) The generalization gap depends on the number of raw features, i.e. d

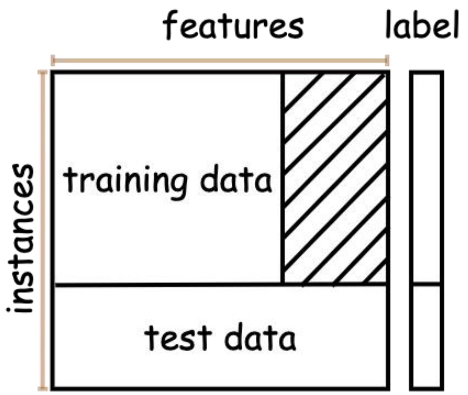
2) The size M is determined by the configuration of proxy training data.

(If there is more randomness, then M would be larger)

Is larger M always better? No! larger variance and larger optimization error

Experiments on UCI Datasets

Dataset	Domain	#Instances	#Raw Feat.	Cardinality	#0-1 Feat.	#Class
Gene	Life	3190	60	4∼6	287	3
Protein	Life	1080	80	2∼8	743	8
Robot	Computer	5456	24	9	237	4
Drive	Computer	58509	49	9	378	11
Calls	Life	7195	10	4∼10	219	10
Github	Social	37700	-	∼	4006	2



- **Evaluation:** training on observed features and testing on all features
  - Instance-level: random split all the instances into training/validation/test data
  - Feature-level: random split all the features into observed/unobserved features
- **Baselines/Competitors:**
  - Base (use observed features for tr/te), Oracle (use all features for tr)
  - Simple extrapolation approaches: Avg, KNN, Mean pooling
  - Incremental learning (first train on observed feat, then retrain on unobserved)
- **Implementation:** 3-layer NN as backbone, 4-layer GNN

Experiments on UCI Datasets

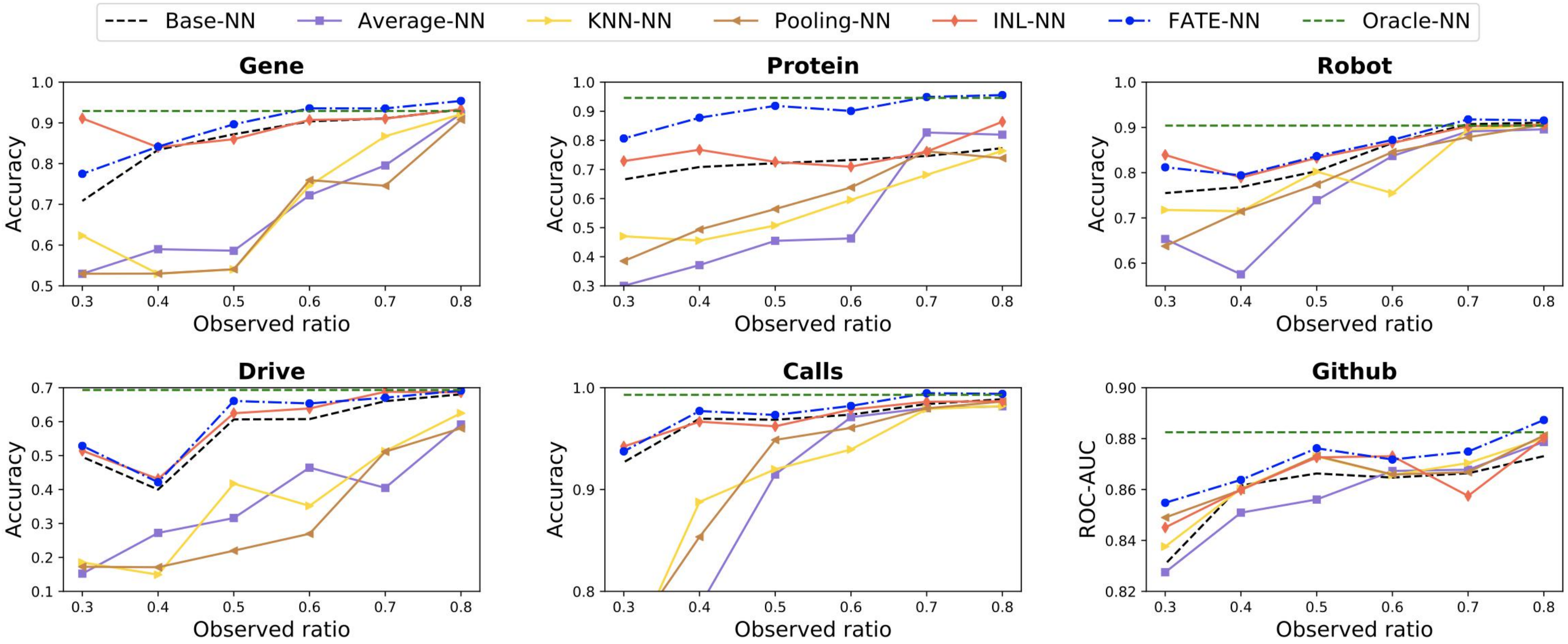


Figure. Accuracy/ROC-AUC results w.r.t. different ratios for observed features

Resutls: 1) FATE (ours) yields 7.3% higher acc. and 1.3% higher AUC than Base

2) FATE achieves very close performance to Oracle (using all features)

2) FATE produces 29.8% higher acc. than baselines Avg, KNN, Pooling

3) FATE even outperforms INL in most cases with averagely 4.1% impv.

Experiments on UCI Datasets

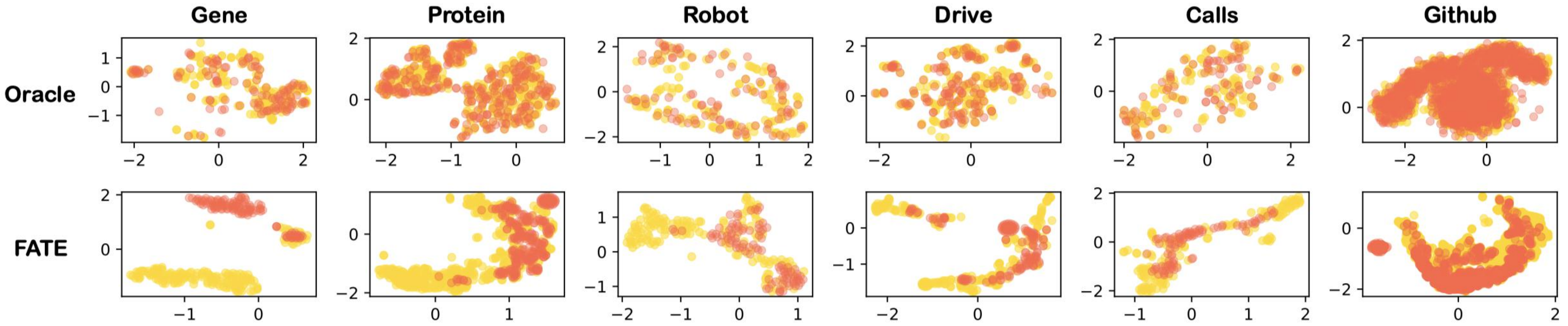


Figure. T-SNE visualization of feature embeddings produced by FATE (ours) and Oracle. Red for observed features and yellow for unobserved ones.

Key insights: 1) FATE's produced embeddings for observed/unobserved features have dissimilar distributions compared to Oracle

➡ *FATE manages to extract some informative knowledge from new features*

2) The embeddings of FATE form some particular structures

➡ *FATE could further capture feature-level relations through GNN interaction*

Experiments on Advertisement Click Prediction

- **Scenario:** click-through rate (CTR) prediction for online advertisement
  - Goal: predict whether a user would click on a displayed ad. (binary classification)
  - Input: attribute features for users/ads
    - Typical features: device id, site id, app id, ad category, app category, etc.
    - The ID features have massive values which induces large feature dimensions

Dataset	Domain	#Instances	#Raw Feat.	Cardinality	#0-1 Feat.	#Class
Avazu	Ad.	40,428,967	22	5~1611749	2,018,025	2
Criteo	Ad.	45,840,617	39	5~541311	2,647,481	2

- **Evaluation:** chronologically split all the instances into 10-fold
  - Use first subset for training, second for validation and the remaining for test
  - ~1.3M/~0.4M/~0.8M exclusive features in training/validation/test data in Criteo
- **Implementation:** 3-layer NN/DeepFM as backbones

Experiments on Advertisement Click Prediction

Table. ROC-AUC results for eight test sets (T1 - T8) on Avazu and Criteo

Dataset	Backbone	Model	T1	T2	T3	T4	T5	T6	T7	T8	Overall
Avazu	NN	Base	0.666	0.680	0.691	0.694	0.699	0.703	0.705	0.705	0.693 ± 0.012
		Pooling	0.655	0.671	0.683	0.683	0.689	0.694	0.697	0.697	0.684 ± 0.011
		FATE	**0.689**	**0.699**	**0.708**	**0.710**	**0.715**	**0.720**	**0.721**	**0.721**	$\mathbf{0.710} \pm 0.010$
	DeepFM	Base	0.675	0.684	0.694	0.697	0.699	0.706	0.708	0.706	0.697 ± 0.009
		Pooling	0.666	0.676	0.685	0.685	0.688	0.693	0.694	0.694	0.685 ± 0.009
		FATE	**0.692**	**0.702**	**0.711**	**0.714**	**0.718**	**0.722**	**0.724**	**0.724**	$\mathbf{0.713} \pm 0.010$
Criteo	NN	Base	0.761	0.761	0.763	0.763	0.765	0.766	0.766	0.766	0.764 ± 0.002
		Pooling	0.761	0.762	0.764	0.763	0.766	0.767	0.768	0.768	0.765 ± 0.001
		FATE	**0.770**	**0.769**	**0.771**	**0.772**	**0.773**	**0.774**	**0.774**	**0.774**	$\mathbf{0.772} \pm 0.001$
	DeepFM	Base	0.772	0.771	0.772	0.772	0.774	0.774	0.774	0.774	0.773 ± 0.001
		Pooling	0.772	0.772	0.773	0.774	0.776	0.776	0.776	0.776	0.774 ± 0.002
		FATE	**0.781**	**0.780**	**0.782**	**0.782**	**0.784**	**0.784**	**0.784**	**0.784**	$\mathbf{0.783} \pm 0.001$

Resutls: FATE achieves significantly improvements over Base/Pooling with different backbones (NN and DeepFM[1])

→ *FATE can extrapolate for unseen features in dynamic data*

[1] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. Deepfm: A factorization-machine based neural network for CTR prediction. In International Joint Conference on Artificial Intelligence, 2017.

Scalability Test for Large Datasets

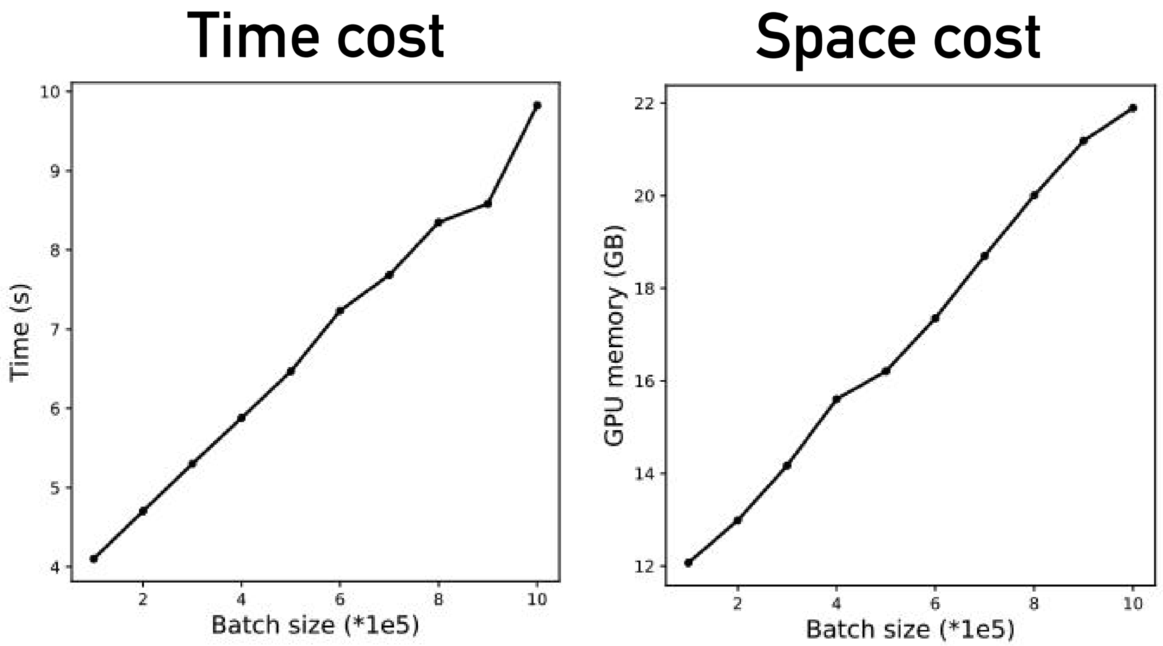


Figure 1. Scalability w.r.t. batch sizes

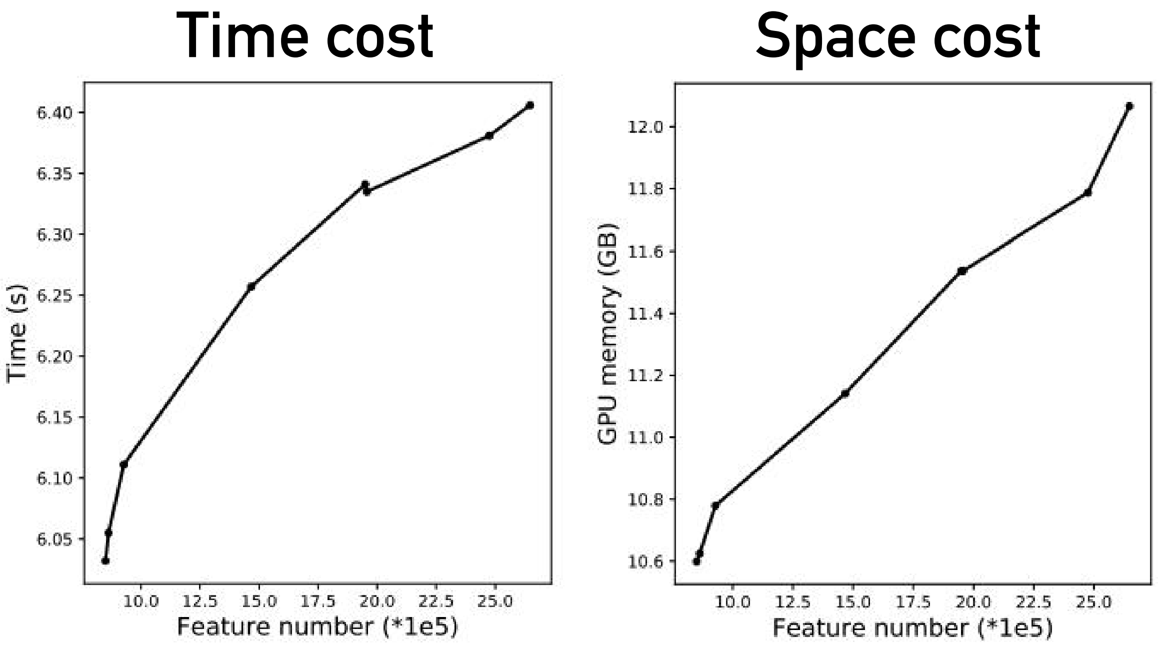


Figure 2. Scalability w.r.t. feature numbers

Resutls: FATE yields linear time/space scalability w.r.t. data and feature sizes

➡ *Promising for larger datasets and real systems*

The feature-data graph representation and GNN learning induces complexity $O(Bd)$

Comparison with Other Learning Problems

- **Domain Adaption:** source domain → target domain
  - Our differences: 1) same label distribution , 2) one task with different input feature space
- **Continual Learning:** data 1 → data 2 → data 3 → data 4 → ···
  - Our differences: 1) one-piece data, 2) no further re-training, 3) one task for learning
- **Open-Set Learning:** training label space → new label class
  - Our differences: 1) same label space, 2) different input feature space
- **Zero-Shot Learning:** observed entities → unseen entities
  - Our differences: 1) no extra side information, 2) different feature space

Conclusions

- Our contributions: new problem setting + new method
  - Formulate the problem of open-world feature extrapolation
  - Propose a graph-learning approach with new training techniques
  - Provide theoretical insights on the generalization performance
  - Empirically verify the effectiveness, applicality and scalability of new methods

Thanks for listening!